# Interaction of Architecture and Algorithm in the Domain-based Parallelization of an Unstructured Grid Incompressible Flow Code

**Dinesh K. Kaushik** & **David E. Keyes**
CS Department, Old Dominion University &
ICASE, NASA Langley Research Center

**Barry F. Smith**
MCS Division, Argonne National Laboratory

## Organization of Presentation

- Issues for unstructured grid domain decomposition methods
- Background of FUN3D
- Background of PETSc
- Illustrations of general porting issues
- Summary of serial and parallel performance
- Conclusions

## Solving Unstructured Grid Problems in Parallel: Main Issues

- SPMD parallelization of unstructured grid solvers is complicated by the fact that no two interprocessor data dependency patterns are alike
- The user-provided global ordering may be incompatible with the subdomain-contiguous ordering required for high performance and convenient SPMD coding
- Loss of **regularity** in unstructured grid solvers makes them more **memory** and **integer-op intensive**; nevertheless, a library-based solver should be competitive **in serial** with a legacy solver in terms of memory and execution time

## Implications of the Memory Hierarchy on Computational Efficiency

- Storage/use patterns should follow memory hierarchy
  - **Blocks** for **Registers**
    block storage format for multicomponent systems – saves CPU cycles
  - **Interlaced Data Structures** for **Cache**
    choose

    $$u1, v1, w1, p1, u2, v2, w2, p2, \ldots$$

    in place of

    $$u1, u2, \ldots, v1, v2, \ldots, w1, w2, \ldots, p1, p2, \ldots$$

  - **Subdomains** for **Distributed Memory**
    "chunky" domain decomposition for optimal surface-to-volume (communication-to-computation) ratio
- This hierarchy is concerned with different issues than the **algorithmic efficiency** issues associated with hierarchies of grids

## Optimal Granularity of Decomposition

For **cache-based microprocessors**, granularity is determined by three forces:

- **Convergence Rate**
  usually deteriorates with increased granularity
- **Communication Volume**
  increases with increased granularity
- **Size of Local Working Set**
  fits better into successively smaller cache levels with increased granularity

## Description of the Legacy Code - FUN3D

- FUN3D is a tetrahedral vertex-centered unstructured grid code developed by W. K. Anderson (LaRC) for compressible and incompressible Euler and Navier-Stokes equations
- Parallel experience is with incompressible Euler so far, but nothing in the algorithms or software changes for the other cases; only convergence rate will vary with conditioning, as determined by Mach and Reynolds numbers (and mesh)
- FUN3D uses 1st- or 2nd-order Roe for convection and Galerkin for diffusion, and false timestepping with backwards Euler for nonlinear continuation towards steady state
- Solver is Newton-Krylov-Schwarz; timestep is advanced towards infinity by the switched evolution/relaxation (SER) heuristic of Van Leer & Mulder

# PETSc —
# a Portable Extensible Toolkit for Scientific Computing

- Gives relatively high-level expression to preconditioned iterative linear solvers, and Newton iterative methods
- Supports complex arithmetic
- Ports wherever MPI ports; committed to progressive MPI tuning
- Permits great flexibility (through object-oriented philosophy) for algorithmic innovation
- Freely available
- Callable from FORTRAN77, C, and C++; written in C
- Includes diagnostic, monitoring, and visualization GUIs

## The PETSc Philosophy

- Library approach — compiler can't do all; users shouldn't do all more than once
- Distributed data structures as fundamental objects — index sets, vectors, and matrices (gridfunctions coming)
- Iterative linear and nonlinear solvers, combinable modularly and recursively, and extensible
- Portable
- Uniform Application Programmer Interface (API)
- Multi-layered entry
- Message-passing detail suppressed

## Conversion of Legacy FUN3D into PETSc/MPI version

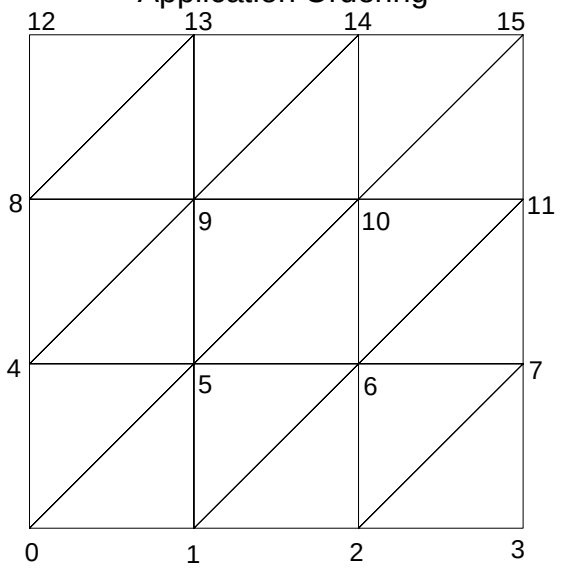
- Project begun 10/96, completed 3/97, undergoing continual enhancement
- Five-month (part-time) effort included:
  - learning FUN3D and the PUNS3D mesh preprocessor
  - learning the MeTiS partitioner
  - adding and testing new functionality in PETSc
  - restructuring FUN3D from vector to cache orientation
- Approximately 3,300 of 14,400 F77 lines of FUN3D retained (primarily as "node code" for flux and Jacobian evaluations); PETSc solvers used for the rest
- Effort has not yet included:
  - Parallel I/O and post-processing
- Next unstructured mesh code port should require significantly less time

## Solving Unstructured Grid Problems in Parallel: Basic Outline of the Solution Strategy

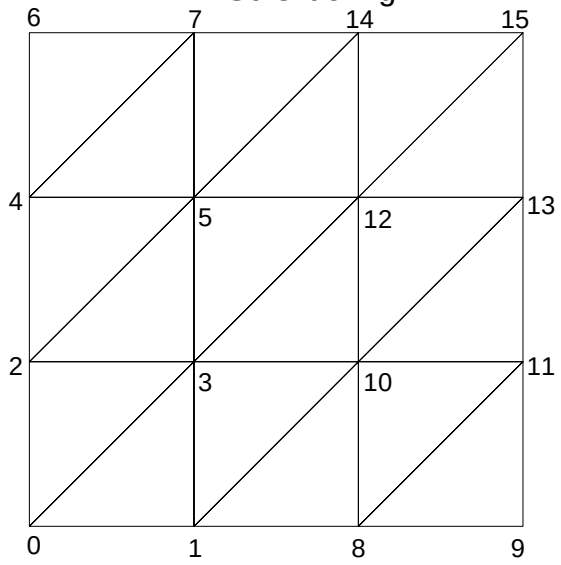
- Follow the "owner computes" rule under the dual constraints of minimizing the number of messages and overlapping communication with computation
- Each processor "ghosts" its stencil dependences in its neighbors
- Ghost nodes ordered after contiguous owned nodes
- Domain mapped from (user) global ordering into local orderings
- Scatter/gather operations created between **local sequential** vectors and **global distributed** vectors, based on runtime connectivity patterns
- Newton-Krylov-Schwarz operations translated into local tasks and communication tasks (nonblocking for overlap where hardware supports)

# Three Different Orderings - In Focus

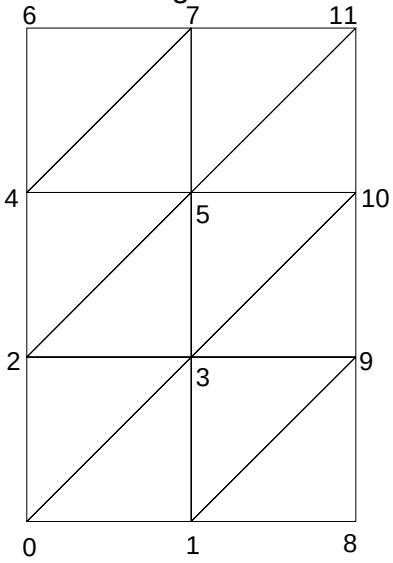
## Application Ordering



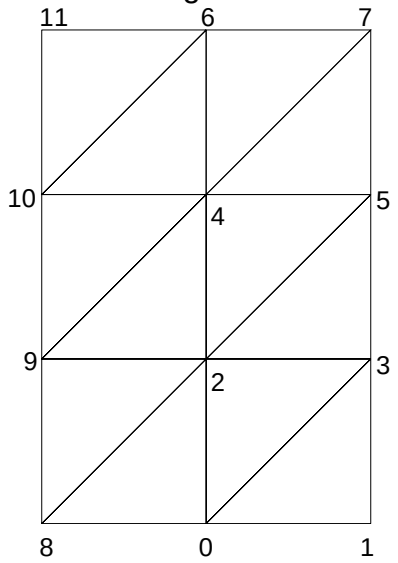

## PETSc Ordering




## Local Ordering for Processor 0




## Local Ordering for Processor 1



## Scattering Between the Orderings

- After establishing different orderings, establish the "scatter" between the global and local vectors in the following way :

```
ISCreateStride(MPI_COMM_SELF,bs*nvertices,0,1,&islocal);
ISCreateBlock(MPI_COMM_SELF,bs,nvertices,svertices,&isglobal);
VecScatterCreate(x,isglobal,user.localX,islocal,&user.scatter);
```

- Next, before using the local vector in any subroutine, carry out the scatter operation :

```
VecScatterBegin(X,localX,INSERT_VALUES,SCATTER_FORWARD,scatter);
VecScatterEnd(X,localX,INSERT_VALUES,SCATTER_FORWARD,scatter);
```

## Sample Serial Performance Comparison: PETSc vs. Legacy Code

For both codes

- same optimization level (-O3) was used
- same timer was used
- time measurement started after reading all the input files
- no output was written during timing measurements
- platform used was IBM SP at Argonne with enough memory to avoid page faults after loading

| | Execution (s) | | Memory (MB) | |
|---|---|---|---|---|
| vert | orignal | PETSc | original | PETSc |
| 2800 | 122.71 | 27.88 | 10.22 | 12.08 |
| 22700 | 2905.30 | 381.09 | 74.74 | 83.67 |

*Percentage difference in memory requirement reduces with problem size*

## Sample Memory Conservation Techniques And Successive Effects in FUN3D Porting History

- Precisely sized preallocation of sparse matrix objects
  (77 $\rightarrow$ 47 MB of RAM)
- Pruning of legacy code solver data structures
  (47 $\rightarrow$ 34 MB of RAM)
- In-place factorization of preconditioner
  (34 $\rightarrow$ 21 MB of RAM)
- Moving "MatSetValues" calls into legacy subroutines
  (21 $\rightarrow$ 16 MB of RAM)
- Making Partitioning Stage Scalable
  (16 $\rightarrow$ 12 MB of RAM)
- Size of legacy code on same problem: 10 MB
- Size of parallel single-node code on same problem: 12 MB

## Summary of Parallel Performance on Cray T3E and IBM SP

- 1.4 million degree-of-freedom problem converged to machine precision in approximately 6 minutes with approximately 1600 flux balance operations (work units) on 128 processors of a T3E or 80 processor of an SP
- Relative efficiencies of 75% to 85% over this range
- Algorithmic efficiency (ratio of iteration count of less decomposed grid to more decomposed grid – using the "best" algorithm for each processor granularity) is in excess of 90% over this range; iteration count is only weakly dependent upon granularity
- Implementation efficiency (ratio of the cost per vertex per iteration) is in excess of **80%** over this range and can be superunitary
- Superunitary implementation efficiency derives from improved cache locality at higher granularity (smaller workingsets on each processor), in spite of greater nearest neighbor communication volume
- Properly sizing workingset to cache largely overcomes convergence and communication penalties of concurrency

## Cray T3E Scalability – Fixed Size

FUN3D-PETSc M6 Wing Test Case, Incompressible Euler
2nd-order Roe Scheme, 1-layer Halo
Tetrahedral grid of 357,900 vertices (1,431,600 unknowns)

| procs | its | exe | speedup | $\eta_{alg}$ | $\eta_{impl}$ | $\eta_{overall}$ |
|---|---|---|---|---|---|---|
| 16 | 77 | 2587.95s | 1.00 | 1.00 | 1.00 | 1.00 |
| 24 | 78 | 1792.34s | 1.44 | 0.99 | 0.97 | 0.96 |
| 32 | 75 | 1262.01s | 2.05 | 1.03 | 1.00 | 1.03 |
| 40 | 75 | 1043.55s | 2.48 | 1.03 | 0.97 | 0.99 |
| 48 | 76 | 885.91s | 2.92 | 1.01 | 0.96 | 0.97 |
| 64 | 75 | 662.06s | 3.91 | 1.03 | 0.95 | 0.98 |
| 80 | 78 | 559.93s | 4.62 | 0.99 | 0.94 | 0.92 |
| 96 | 79 | 491.40s | 5.27 | 0.97 | 0.90 | 0.88 |
| 128 | 82 | 382.30s | 6.77 | 0.94 | 0.90 | 0.85 |

*85% relative efficiency at 128 nodes*

## IBM SP Scalability – Fixed Size

FUN3D-PETSc M6 Wing Test Case, Incompressible Euler
2nd-order Roe Scheme, 1-layer Halo
Tetrahedral grid of 357,900 vertices (1,431,600 unknowns)

| procs | its | exe | speedup | $\eta_{alg}$ | $\eta_{impl}$ | $\eta_{overall}$ |
|---|---|---|---|---|---|---|
| 8 | 70 | 2897.46s | 1.00 | 1.00 | 1.00 | 1.00 |
| 10 | 73 | 2405.66s | 1.20 | 0.96 | 1.00 | 0.96 |
| 16 | 78 | 1670.67s | 1.73 | 0.90 | 0.97 | 0.87 |
| 20 | 73 | 1233.06s | 2.35 | 0.96 | 0.98 | 0.94 |
| 32 | 74 | 797.46s | 3.63 | 0.95 | 0.96 | 0.91 |
| 40 | 75 | 672.90s | 4.31 | 0.93 | 0.92 | 0.86 |
| 48 | 75 | 569.94s | 5.08 | 0.93 | 0.91 | 0.85 |
| 64 | 74 | 437.72s | 6.62 | 0.95 | 0.87 | 0.83 |
| 80 | 77 | 386.83s | 7.49 | 0.91 | 0.82 | 0.75 |

*75% relative efficiency at 80 nodes*

## Cray T3E Scalability – Gustafson

FUN3D-PETSc M6 Wing Test Case, Incompressible Euler
2nd-order Roe Scheme, 1-layer Halo
Tetrahedral grid

| vert | procs | vert/proc | its | exe | exe/it |
|---|---|---|---|---|---|
| 357,900 | 80 | 4474 | 78 | 559.93s | 7.18s |
| 53,961 | 12 | 4497 | 36 | 265.72s | 7.38s |
| 9,428 | 2 | 4714 | 19 | 131.07s | 6.89s |

*Less than 7% variation in performance*
*over factor of nearly 40 in problem size*

## Notes on Efficiency

Conflicting definitions of parallel efficiency abound, depending upon two choices:

- What scaling is to be used as the number of processors is varied?
  - overall fixed-size problem
  - varying size problem with fixed memory per processor
  - varying size problem with fixed work per processor
- What form of the algorithm is to be used as number of processor is varied?
  - reproduce the sequential arithmetic exactly
  - adjust parameters to perform best on each given number of processors

Our charts include both overall fixed-size scaling and approximately fixed memory per processor (Gustafson) scaling

We always adjust the subdomain blocking parameter to match the number of processors, one subdomain per processor; this causes the number of iterations to vary

## Notes on Efficiency, cont.

Effect of changing-strength preconditioner and effect of parallel overhead are often separated into algorithmic and implementation factors

- Customary definition of overall efficiency in going from $q$ to $p$ processors ($p > q$):

$$\eta(p|q) = \frac{q \cdot T(q)}{p \cdot T(p)}$$

  where $T(p)$ is the overall execution time on $p$ processors (measured)

- Factor $T(p)$ into $I(p)$, the number of iterations, and $C(p)$, the average cost per iteration.

- Algorithmic efficiency is measure of preconditioning quality (measured):

$$\eta_{alg}(p|q) = \frac{I(q)}{I(p)}$$

- Implementation efficiency is remaining (inferred, not directly measurable) factor:

$$\eta_{impl}(p|q) = \frac{q \cdot C(q)}{p \cdot C(p)}$$

## Footnotes on Scalability Tables

- "its" represents the number of pseudo-transient Newton steps — one Newton step per timestep, with SER growth in timestep up to a CFL of 100,000, and with a maximum number (20) of Schwarz-preconditioned GMRES steps per Newton step with relative tolerance of $10^{-2}$
- Convergence defined as a relative reduction in the norm of the steady-state nonlinear residual by a factor of $10^{-10}$
- Convergence rate typically degrades slightly as number of processors is increased, due to introduction of concurrency in preconditioner — highly partition-dependent
- Implementation efficiency may improve slightly as processors are added, due to smaller workingsets — better cache residency
- Implementation efficiency ultimately degrades as communication-to-computation ratio increases

## Our View of the "State-of-the-Art": Architecture and Programming Environment

- Vector-awareness is **out**; cache-awareness is **in**; vector-awareness **will return** in subtle ways
- Except for Tera and installed vector base, all near-term large-scale computers will be based on commodity processors
- HPF and parallel compilers not yet up to performance
- Some useful parallel libraries, like PETSc
- Need for better memory bandwidth to harness the full capability of future (& current) chips

## Our View of the "State-of-the-Art": Algorithms

- Explicit time integration is solved problem, except for dynamic mesh adaptivity
- Implicit methods remain a major challenge:
  - Today's algorithms leave something to be desired in convergence rate
  - All good algorithms have global synchronization
- Data parallelism from domain decomposition is unquestionably the main source of locality-preserving concurrency, but good smoothers and preconditioners violate locality
- New forms of **algorithmic** latency tolerance must be found
- Exotic methods should be considered at ASCI scales

## Our View of the "State-of-the-Art": Application-Algorithm-Architecture Interaction

- Ripest remaining advances are interdisciplinary
- Application-Algorithm
  - Ordering, partitioning, and coarsening must adapt to coefficients (grid spacing and flow magnitude and direction)
  - Trade-offs between pseudo-time iteration, nonlinear iteration, linear iteration, and preconditioner iteration must be understood and exploited
- Algorithm-Architecture
  - Algorithmicists must think natively in parallel and avoid introducing unnecessary sequential constraints
  - Algorithmicists should inform their choices with what their machine is good at and what it is bad at

## Conclusions

- Hierarchy of domain decomposition should follow distributed memory hierarchy for computational efficiency
  - **blocking for registers**
    gives a factor of 2 in performance for multicomponent systems
  - **interlaced data structure for cache**
    reduces execution time by more than a factor of 4
  - **subdomains for processor memory**
    migrates the sequential code to SPMD parallelism
- In addition to **convergence rate** and **communication volume**, **working set size** is another parameter to consider for "preferred" granularity of domain decomposition
- PETSc ported FUN3D gives nice scalability results (**parallel efficiency ranges from 75%–85%**) on two platforms - IBM SP and Cray T3E
- Library (PETSc) based solver is **competitive** with the legacy solver – outperforms by a factor of 9 even in **serial** mode – percentage difference in memory reduces with problem size

## Reference URLs

- FUN3D
  http://fmad-www.larc.nasa.gov/~wanderso/Fun/fun.html
- PETSc
  http://www.mcs.anl.gov/petsc/petsc.html
- Pointers and related papers
  http://www.cs.odu.edu/~keyes/keyes.html